LOGICS FOR UNRANKED TREES

Leonid Libkin

University of Toronto

Paper in the proceedings

- A much shortened version of the survey.
- Full version on my webpage
  - google.com $\Rightarrow$ `libkin` $\Rightarrow$ "I'm feeling lucky"
- Why? Limits on the number of pages in a single volume.

Paper in the proceedings

- A much shortened version of the survey.
- Full version on my webpage
  - google.com $\Rightarrow$ `libkin` $\Rightarrow$ "I'm feeling lucky"
- Why? Limits on the number of pages in a single volume.
- Apparently there is no Moore's Law in book binding technology:
  - Gutenberg's Bible, published in 1455 — 1282 pages
  - Springer's ICALP Proceedings, published in 2005 – 1477 pages

Trees are everywhere

- One of the most common objects we see in CS.
- Logics and automata on trees found many applications:
  - verification;
  - program analysis;
  - logic programming/constraint programming;
  - linguistics;
  - databases

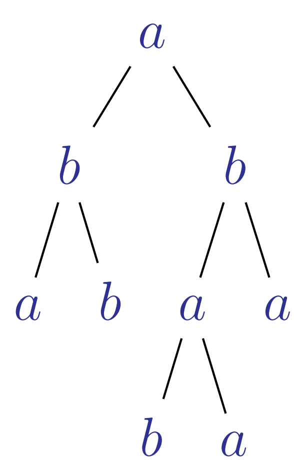
Logic/automata connection

- Regular tree languages = given by tree automata.
- Typically characterized via MSO — Monadic Second-Order logic
  - MSO is an extension of first-order logic with quantification over sets.
- Classical results (about 35 years old):
  - Thatcher-Wright: For finite binary trees, Regular = MS0-definable.
  - Rabin: Same is true for infinite trees. Hence S2S, the MSO-theory of two successors, is decidable.
  - This is one of the most powerful decidability results.
- Many more results followed (Thomas+colleagues, Wilke, Walukiewicz, Segoufin, Schwentick, Neven, etc etc)

Ranked Trees

Typically one works with ranked trees; e.g., binary, ternary, etc trees.

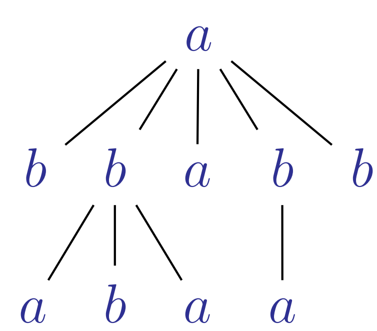
A binary tree:



Unranked trees

Recently focus has shifted towards unranked trees.

In them, nodes can have arbitrarily many children, and different nodes may have different number of children.



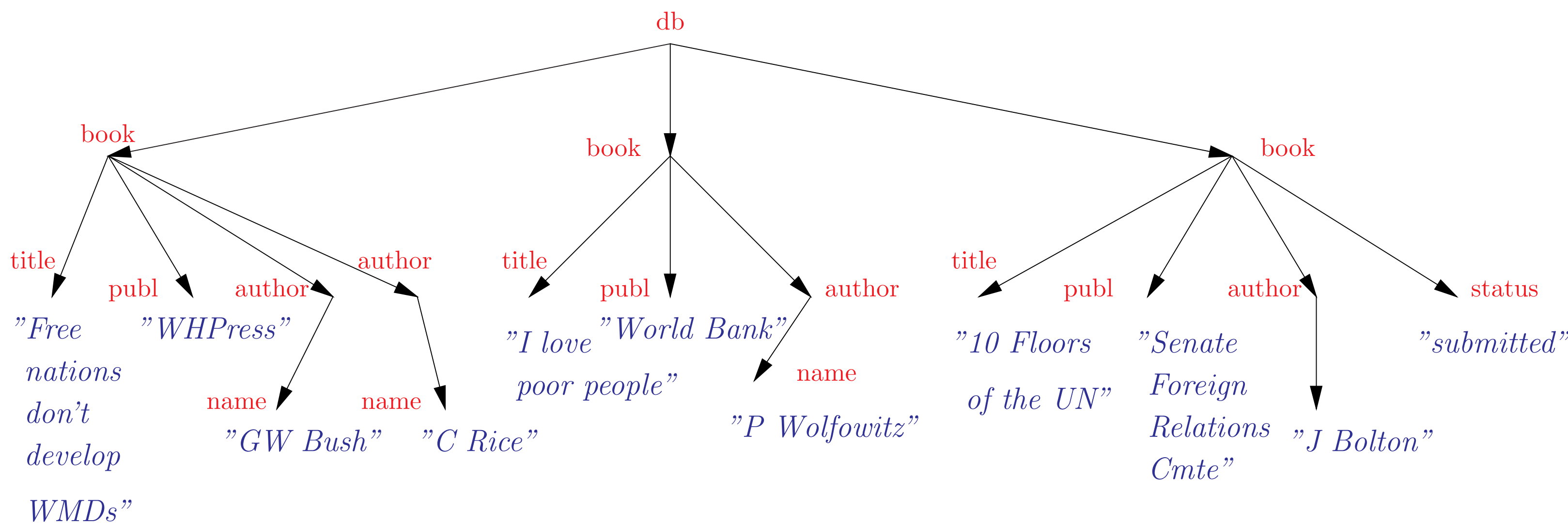
Why unranked trees?

- Main reason: XML.
- XML = eXtensible Markup Language, the standard for exchanging data on the web.
- XML data is modeled as unranked trees.
- A lot of recent work on XML: W3C standards such as XML Schema, XPath, XSLT, XQuery define types, navigation mechanism, transformations, and query languages for XML.
- Active work on XML in many communities, especially databases, information retrieval.
- Brings techniques (sometimes old and almost forgotten) from formal language theory and merges them nicely with logic.

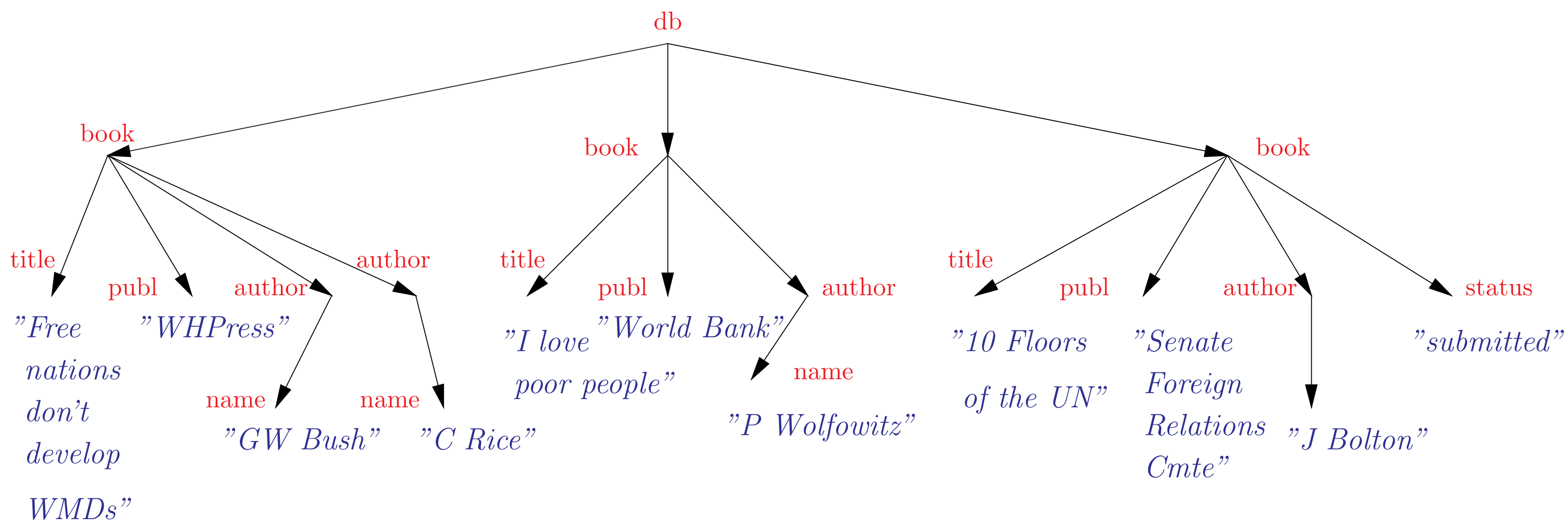
XML documents look like this

```
<db>
  <book>
    <title attr_title="Free Nations don't develop WMD"></title>
    <publisher publ_attr="White House Press"></publisher>
    <author>
       <name name_attr="GW Bush">
    </author>
    <author>
       <name name_attr="C Rice">
    </author>
  </book>
  <book>
    <title attr_title="I Love Poor People"></title>
    <publisher publ_attr="World Bank Press"></publisher>
    <author>
       <name name_attr="P Wolfowitz">
    </author>
  </book>
  <book>
    <title attr_title="10 Floors of the United Naion"></title>
    <publisher publ_attr="Senate Foreign relations Committee"></publisher>
    <status status_attr="submitted"></status>
    <author>
       <name name_attr="J Bolton">
    </author>
  </book>
</db>
```

But we like to view them as unranked trees:



But we like to view them as unranked trees:



Document description (DTD = Document Type Definition)

$$\begin{aligned} \text{db} &\longrightarrow \text{book}^* \\ \text{book} &\longrightarrow \text{title}, \text{publ}, \text{author}^+, \text{status}? \\ \text{author} &\longrightarrow \text{name} \end{aligned}$$

plus attribute names.

Why are we interested in logics?

- XML documents describe data.
- Standard relational database approach:
  - data model – relations
  - declarative languages for specifying queries
  - procedural languages for evaluating queries
- Standard declarative languages are all logic-based:
  - relational calculus = first-order logic (FO)
  - datalog = fragment of fixed-point logic
  - basic SQL = FO with counting

What does XML add?

- New logics.
- New procedural languages:
  - logic–automata connection.

What do logics do?

Most commonly they define:

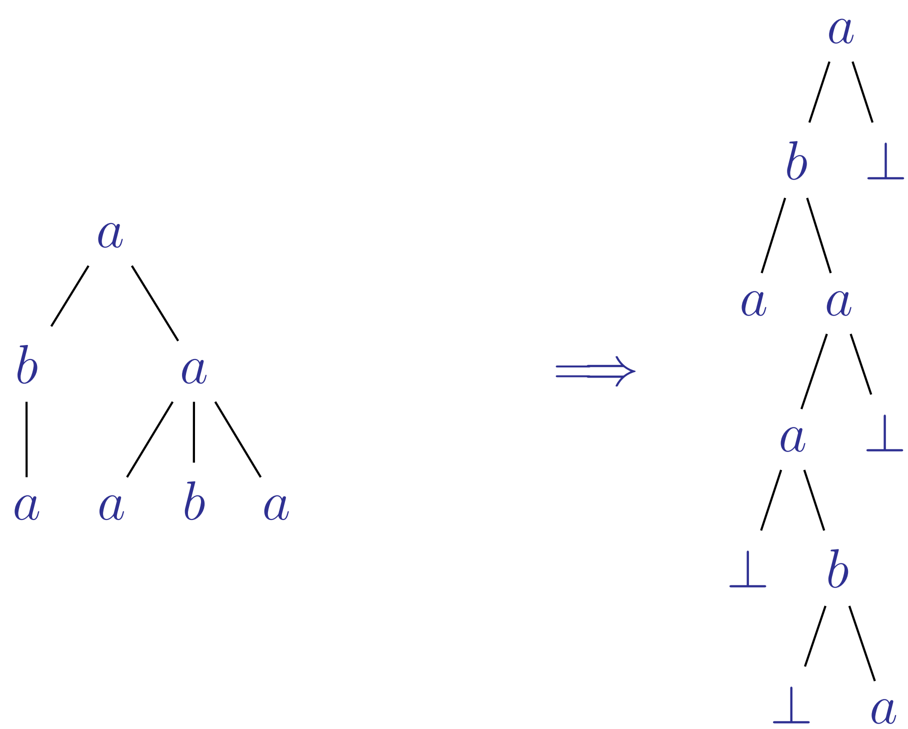
- Boolean (that is, yes/no) queries:
  - DTD conformance
  - Existence of certain paths
- Unary queries which select nodes in trees:
  - all nodes reachable by a certain path from the root;
  - all nodes with a certain label or certain data element.

Commonalities between logics

- (Almost) all have associated automata models.
- Crucial aspect is navigation.
- Hence we often see close connection with temporal logics.
- Logics are specifically designed for unranked trees.

Ranked/Unranked Connection

(used by Rabin in 1970 to interpret SωS in S2S):



It preserves recognizability by automata, MSO-definability, FO-definability...

Why not just use it?

- Instead of defining logics for unranked trees, just translate them into binary trees and use known logical formalisms.
- Problem: hard to navigate!
- A path in a translation becomes a union of arbitrarily many child paths and sibling paths.
- Hard (at least not natural) to express many properties such as DTD conformance.

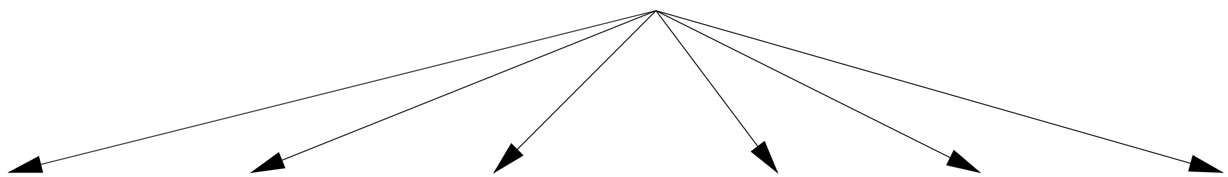
Classification: Yardstick logic

Most logics are based either on FO or MSO.

- FO:
  - Boolean connectives $\vee, \wedge, \neg$,
  - quantifiers $\exists x,\ \ \forall x$ ranging over nodes of trees.
- MSO: in addition
  - quantifiers $\exists X,\ \ \forall X$ ranging over sets of nodes;
  - plus new formulae $x \in X$.

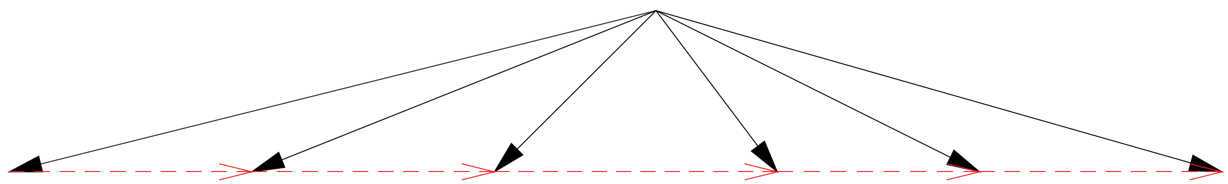
Classification: Ordered vs Unordered Trees

In unordered trees, there is no order among siblings (children of the same node).



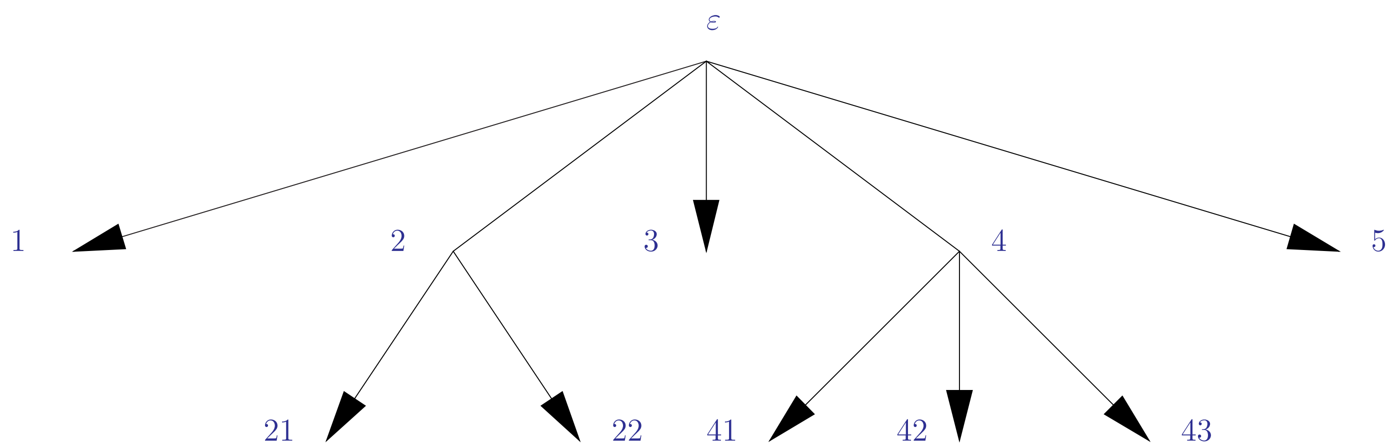
Classification: Ordered vs Unordered Trees

In unordered trees, there is no order among siblings (children of the same node).



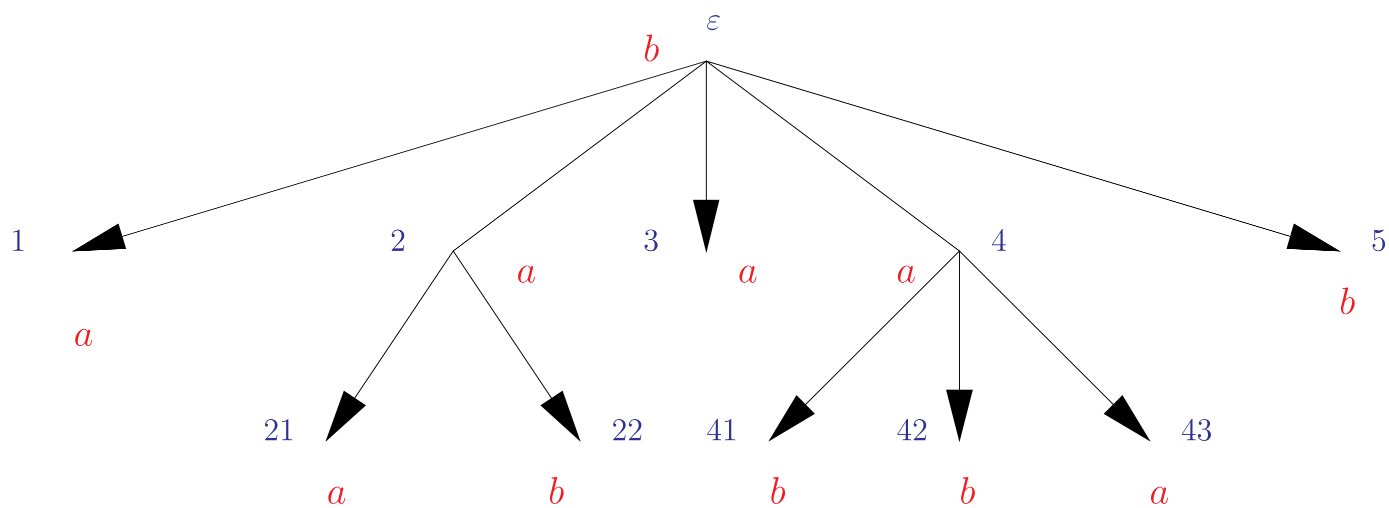
In ordered trees, siblings are ordered (from the oldest to the youngest).

Formal definition of unranked trees



Tree domain: prefix-closed subset D of $\mathbb{N}^*$ such that $s \cdot i \in D$ implies $s \cdot j \in D$ for $j < i$.

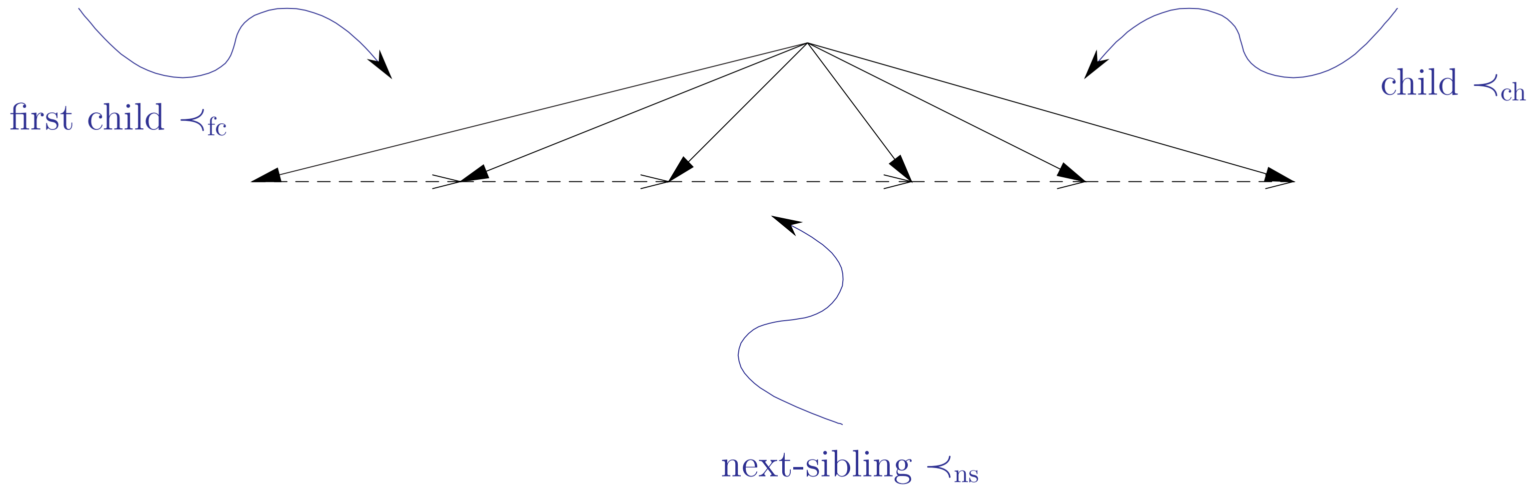
Formal definition of unranked trees



Tree domain: prefix-closed subset D of $\mathbb{N}^*$ such that $s \cdot i \in D$ implies $s \cdot j \in D$ for $j < i$.

Tree over finite alphabet Σ: tree domain plus a mapping from it to Σ.

Basic predicates



Transitive closures:

- $\prec^*_{\mathrm{ch}}$ of $\prec_{\mathrm{ch}}$ (descendant)
- $\prec^*_{\mathrm{ns}}$ of $\prec_{\mathrm{ns}}$ (younger child)

We normally use transitive closures (since they are not definable in FO).

For MSO, we can use either $\prec_{\mathrm{ch}}, \prec_{\mathrm{ns}}$ or $\prec^*_{\mathrm{ch}}, \prec^*_{\mathrm{ns}}$ as they are interdefinable.

LOGICS FOR ORDERED TREES

Logic/automata connection

A set $\mathcal{T}$ of trees is definable in a logic $\mathcal{L}$ iff there is a sentence φ of $\mathcal{L}$ such that

$$T \in \mathcal{T} \quad \Leftrightarrow \quad T \models \varphi$$

A set $\mathcal{T}$ of trees is regular if it is recognizable by a tree automaton.

Theorem

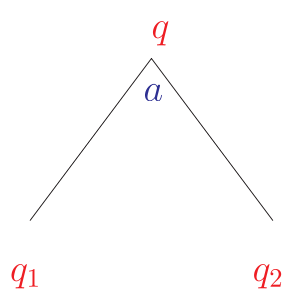
- A set of binary trees is regular iff it is MSO-definable (Thatcher-Wright, 1966).
- A set of unranked trees is regular iff it is MSO-definable (almost folklore; stated many times by different authors).

Tree automata: the ranked case

Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \to 2^{\mathsf{States}}$.

Tree automata: the ranked case

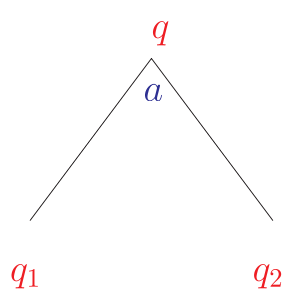
Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.

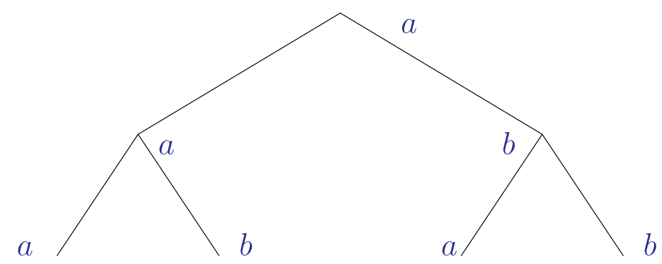


if $q \in \delta(q_1, q_2, a)$

Tree automata: the ranked case

Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.

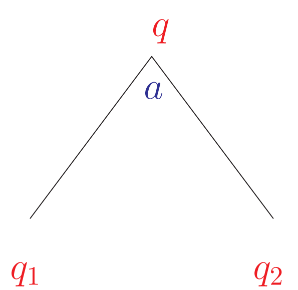




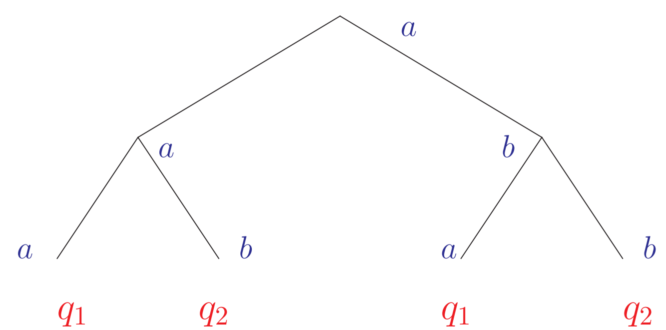
if $q \in \delta(q_1, q_2, a)$

Tree automata: the ranked case

Transitions are δ : States $\times$ States $\times\ \Sigma \longrightarrow 2^{\text{States}}$.

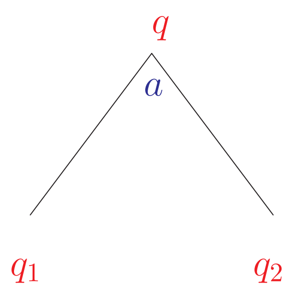


if $q \in \delta(q_1, q_2, a)$

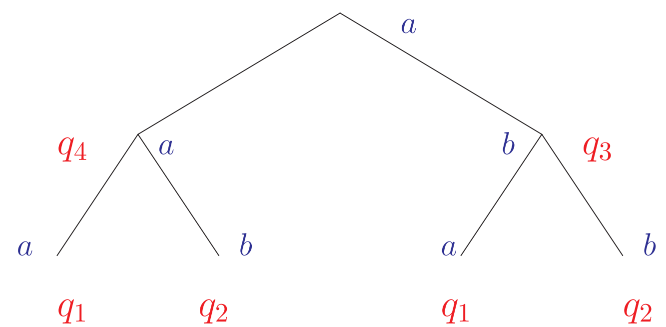


Tree automata: the ranked case

Transitions are δ : States $\times$ States $\times\ \Sigma \longrightarrow 2^{\text{States}}$.

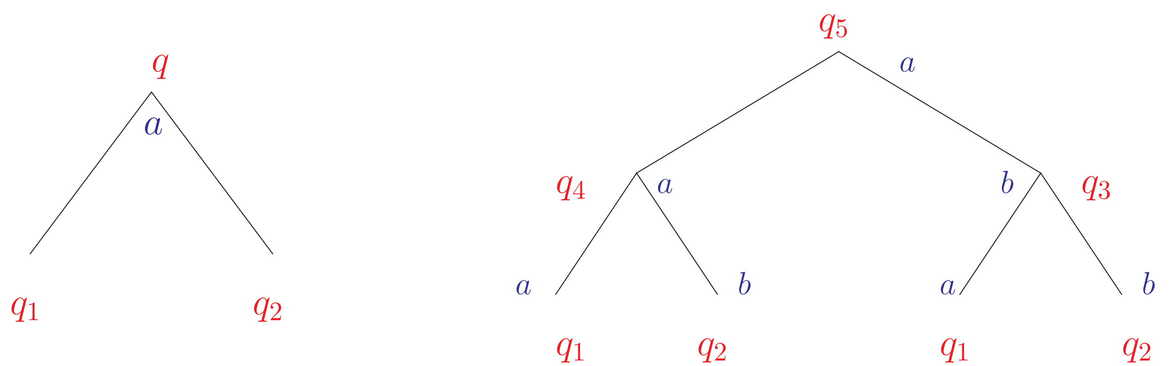


if $q \in \delta(q_1, q_2, a)$



Tree automata: the ranked case

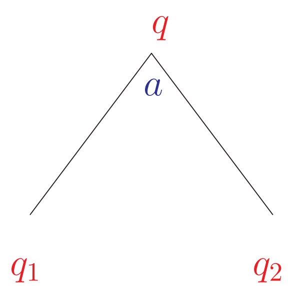
Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.



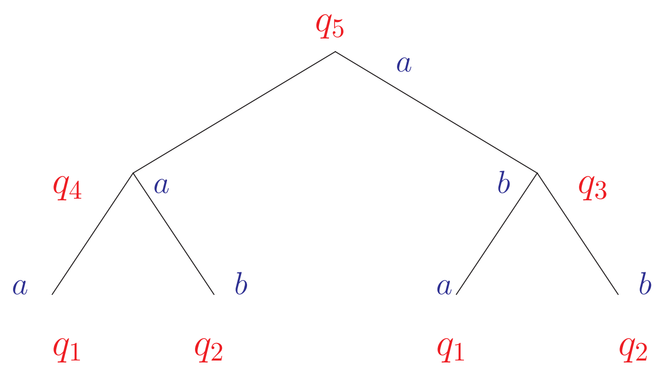
if $q \in \delta(q_1, q_2, a)$

Tree automata: the ranked case

Transitions are $\delta : \mathsf{States} \times \mathsf{States} \times \Sigma \longrightarrow 2^{\mathsf{States}}$.



if $q \in \delta(q_1, q_2, a)$



accepted if q_5 is a final state

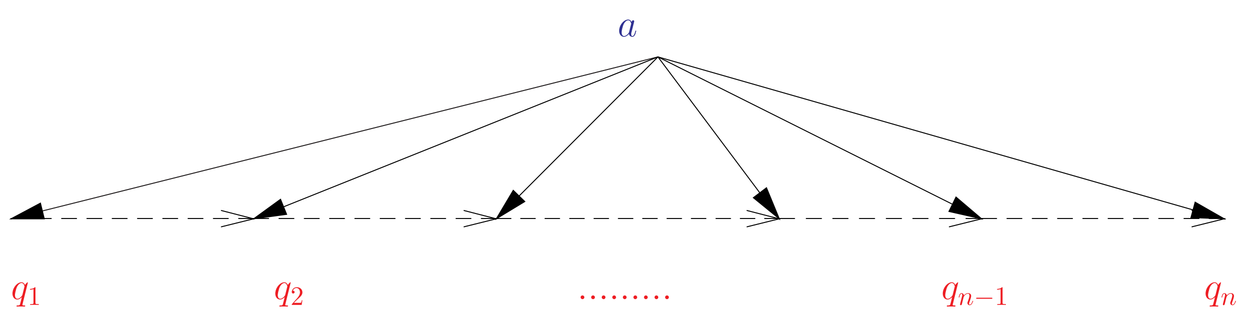
Tree automata: unranked case

Transitions are $\delta : \mathsf{States} \times \Sigma \to 2^{\mathsf{States}^*}$ so that each $\delta(q, a) \subseteq \mathsf{States}^*$ is a regular language.

Tree automata: unranked case

Transitions are $\delta : \mathsf{States} \times \Sigma \to 2^{\mathsf{States}^*}$ so that each $\delta(q, a) \subseteq \mathsf{States}^*$ is a regular language.

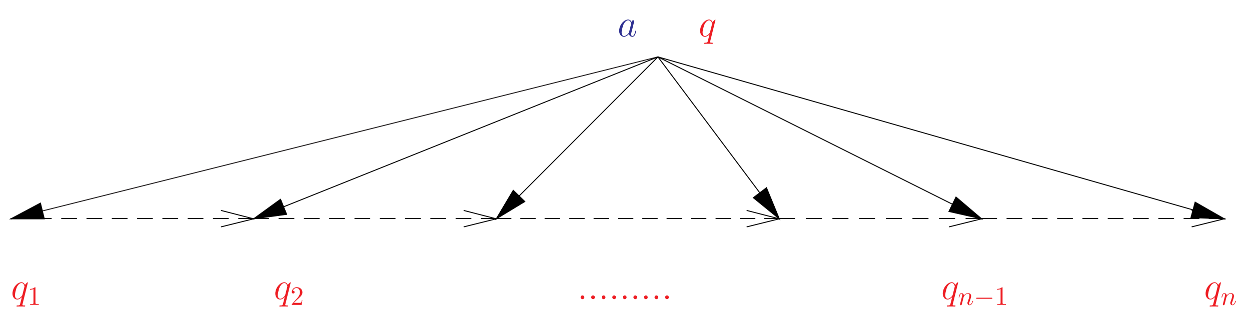
The run is the same as before:



Tree automata: unranked case

Transitions are $\delta : \text{States} \times \Sigma \longrightarrow 2^{\text{States}^*}$ so that each $\delta(q, a) \subseteq \text{States}^*$ is a regular language.

The run is the same as before:

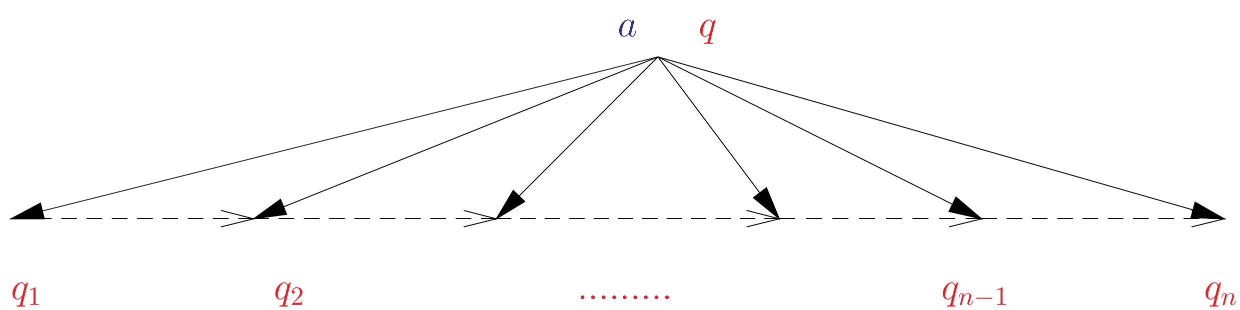


if $q_1 \cdots q_n \in \delta(q, a)$

Tree automata: unranked case

Transitions are $\delta : \mathsf{States} \times \Sigma \to 2^{\mathsf{States}^*}$ so that each $\delta(q,a) \subseteq \mathsf{States}^*$ is a regular language.

The run is the same as before:



if $q_1 \cdots q_n \in \delta(q,a)$

A tree is accepted if there is a run such that the root is assigned an accepting state.

Unary queries

- A unary query selects a set of nodes in a tree.
- A surprisingly simple automaton model captures them.
- Query automaton QA $=$ unranked tree automaton $+$ selecting set $S \subseteq$ States $\times\ \Sigma$.
- QA selects a node s from a tree T if there is a run that assigns a state q to s such that
$$(q, a) \in S.$$

Theorem (Neven/Schwentick, 1999) For unary queries over unranked trees,

$$\text{Query Automata} \quad = \quad \text{MSO}.$$

MSO and DTDs

- Recall that DTDs have rules such as

$$\text{book} \quad \longrightarrow \quad \text{title}, \text{publ}, \text{author}^+, \text{status}?$$

- Since regular string languages are precisely those MSO-definable, it follows that all DTDs are MSO-definable.
- Are DTDs and MSO equal?
- The answer is negative.

MSO and DTDs cont'd

- EDTDs $=$ Extended DTDs: these are DTDs over a larger alphabet $\Sigma' \supseteq \Sigma$ together with a projection $\pi : \Sigma' \rightarrow \Sigma$.
- Trees over Σ that conform to an EDTD: projections of conforming trees over Σ'

Theorem (Thatcher 1967; rediscovered several times recently)

$$\text{EDTDs} \quad = \quad \text{MSO}$$

MSO and DTDs cont'd

- EDTDs = Extended DTDs: these are DTDs over a larger alphabet $\Sigma' \supseteq \Sigma$ together with a projection $\pi : \Sigma' \to \Sigma$.
- Trees over Σ that conform to an EDTD: projections of conforming trees over Σ'

Theorem (Thatcher 1967; rediscovered several times recently)

$$\text{EDTDs} \quad = \quad \text{MSO}$$

- DTDs are not even closed under $\vee$ and $\neg$.
- Unions of DTDs correspond to the existential fragment of MSO over a smaller vocabulary.

MSO over trees: Complexity

- Model-checking problem:

INPUT:	tree T, sentence φ
QUESTION:	Is $T \models \varphi$?

- Two parameters:
  - $\|T\|$ – data complexity
  - $\|\varphi\|$ – query complexity
- By translation to automata: The data complexity of MSO is linear.
- Problem: if we keep data complexity linear, the query complexity is necessarily non-elementary! (Frick/Grohe, 2002)
- Can we do better?
- Yes, by finding different logics that have the power of MSO, and yet better model-checking properties.

Changing syntax to lower complexity: LTL

Syntax:

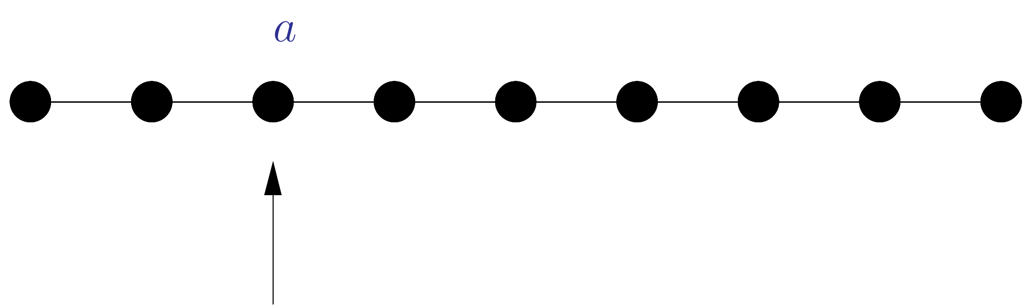
$$\varphi \;:=\; a(\in \Sigma) \;\mid\; \varphi \vee \varphi' \;\mid\; \neg\varphi \;\mid\; \mathbf{X}\varphi \;\mid\; \varphi\mathbf{U}\varphi'$$

Changing syntax to lower complexity: LTL

Syntax:

$$\varphi \;:=\; a(\in \Sigma) \;\mid\; \varphi \vee \varphi' \;\mid\; \neg\varphi \;\mid\; \mathbf{X}\varphi \;\mid\; \varphi \mathbf{U} \varphi'$$

Semantics:



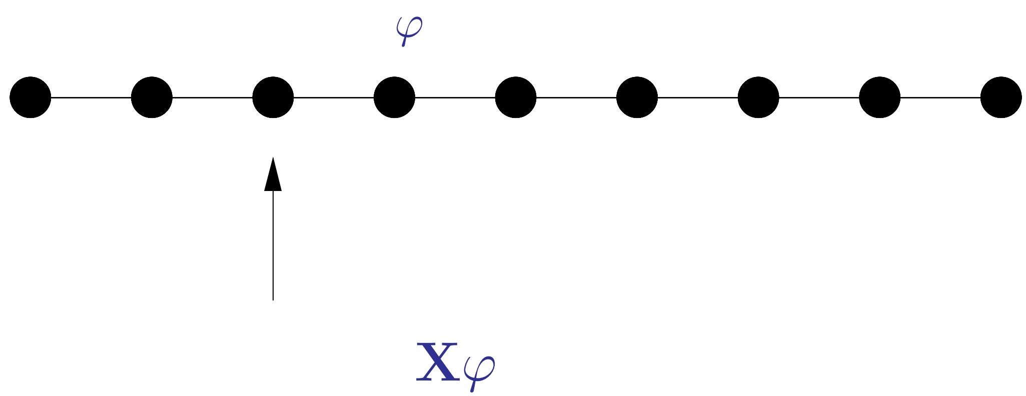
$$a, a \in \Sigma$$

Changing syntax to lower complexity: LTL

Syntax:

$$\varphi \; := \; a(\in \Sigma) \;\mid\; \varphi \vee \varphi' \;\mid\; \neg\varphi \;\mid\; \mathbf{X}\varphi \;\mid\; \varphi\mathbf{U}\varphi'$$

Semantics:

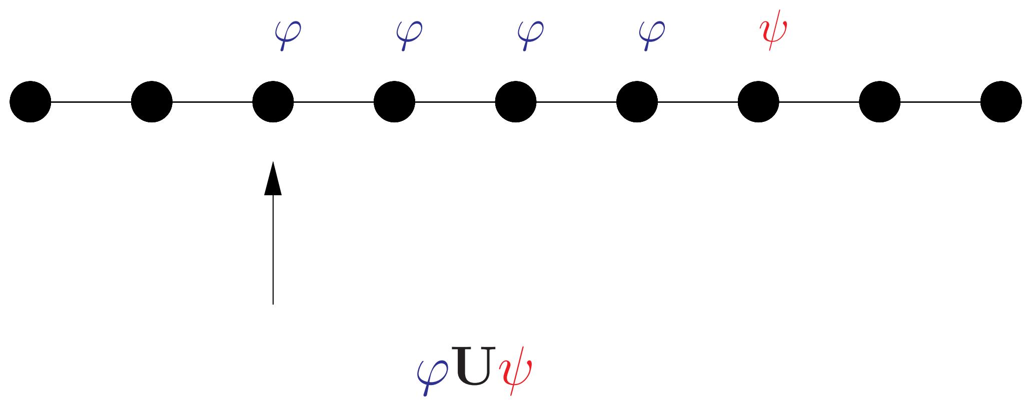


Changing syntax to lower complexity: LTL

Syntax:

$$\varphi \ := \ a(\in \Sigma) \ \mid \ \varphi \vee \varphi' \ \mid \ \neg\varphi \ \mid \ \mathbf{X}\varphi \ \mid \ \varphi\mathbf{U}\varphi'$$

Semantics:



LTL cont'd

- LTL $=$ FO over strings (Kamp's theorem).
- To evaluate LTL with linear data complexity, one needs non-elementary query complexity
(modulo some complexity-theoretic assumptions; Frick/Grohe 2002)
- But LTL over strings can be evaluated in time

$$2^{O(\|\varphi\|)} \cdot |s|$$

- Of course this implies that translation from FO to LTL is non-elementary.

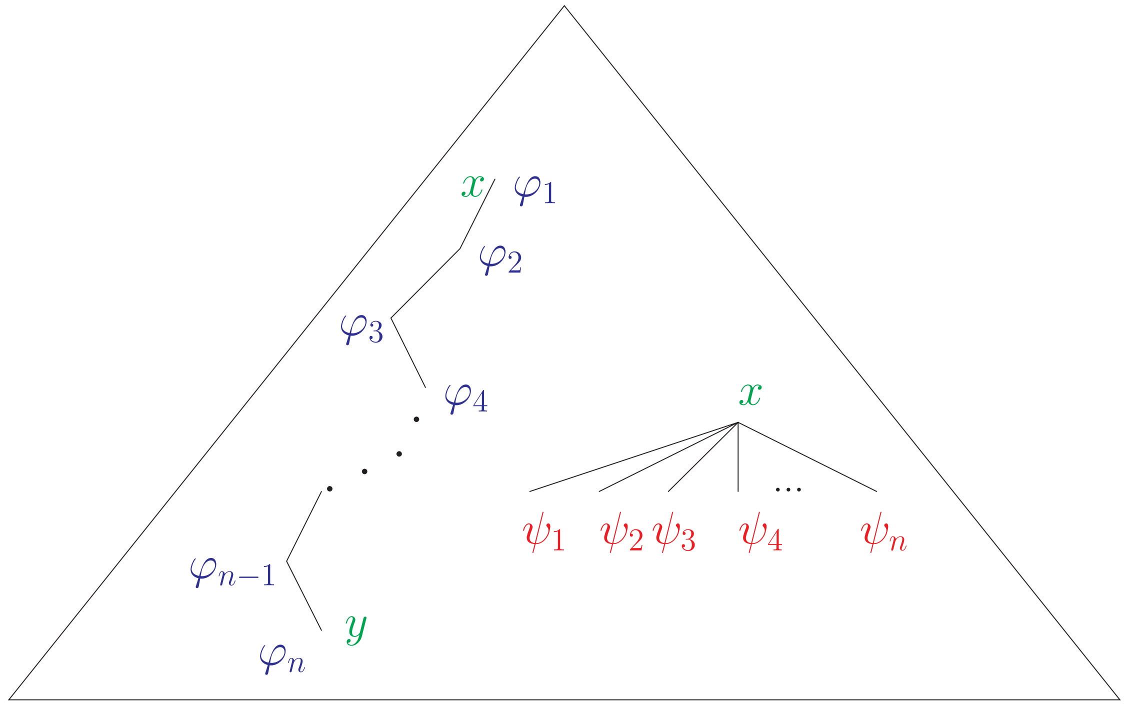
Efficient Tree Logic (ETL)

Neven/Schwentick, 2000

Idea of ETL: take MSO and

- Disallow:
  - next-sibling $\prec_{\mathrm{ns}}$;
  - arbitrary quantification;
- Add:
  - guarded quantification over children or (sets of) descendants;
  - regular expressions over formulae.
  - and put some syntactic restrictions.

Efficient Tree Logic (ETL) cont'd



$\varphi\_e(x, y): \quad \varphi_1 \cdots \varphi_n \in e$ $\qquad$ $\psi\_e(x): \quad \psi_1 \cdots \psi_n \in e$

Efficient Tree Logic (ETL) cont'd

Theorem (Neven/Schwentick)

- ETL $=$ MSO.
- ETL formulae can be evaluated in time

$$2^{O(\|\varphi\|)} \cdot \|T\|$$

Can one do better? – Monadic Datalog

- Datalog $=$ database query language; essentially extension of positive FO with least fixed point.
- Can also be viewed as prolog without function symbols.
- Datalog program is monadic if all introduced predicates (intensional predicates) are monadic – have one free variable.
- Example: select (in predicate D) all nodes s such that all their descendants (including s) are labeled a:

$$
\begin{array}{rcllll}
D(x) & :- & P_a(x), & \mathsf{Leaf}(x) & & \\
D(x) & :- & P_a(x), & x \prec_{\mathrm{fc}} y, & S(y) & \\
S(y) & :- & P_a(x), & \mathsf{LastChild}(y), & D(y) & \\
S(y) & :- & P_a(x), & x \prec_{\mathrm{ns}} y, & S(y), & D(y)
\end{array}
$$

Monadic Datalog cont'd

Assume that Leaf and LastChild are available as basic predicates.

Theorem (Gottlob/Koch, 2002)

- Monadic Datalog $=$ MSO
- A Monadic Datalog program $\mathcal{P}$ can be evaluated on a tree T in time

$$O(\|\mathcal{P}\| \cdot \|T\|)$$

μ-calculus over unranked trees

- μ-calculus (Kozen 82): extension of a temporal logic with the least fixed point operator.
- Subsumes many logics used in verification: LTL, CTL, CTL*.
- Syntax:

$$\varphi \ := \ S \mid a \mid \varphi \vee \varphi' \mid \varphi \wedge \varphi' \mid \neg\varphi \mid \mathbf{X}_E\varphi \mid \mu S\ \varphi(S)$$

  - S must occur positively;
  - E ranges over relations $\prec_{\text{ch}}$ and $\prec_{\text{ns}}$
- Full μ-calculus: one can talk about the past.
  - That is, E also ranges over inverses of $\prec_{\text{ch}}$ and $\prec_{\text{ns}}$

μ-calculus over unranked trees cont'd

Theorem (Barcelo, L., '05) Over unranked trees,

$$\text{full } \mu\text{-calculus} \quad = \quad \text{MSO}$$

For Boolean queries:

- MSO = alternation-free μ-calculus (all negations pushed to atomic formulae)
- Complexity of model-checking (Mateescu, '02):
  - $O(\|\varphi\|^2 \cdot \|T\|)$ for μ-calculus;
  - $O(\|\varphi\| \cdot \|T\|)$ for alternation-free μ-calculus.

Remark: it is well known that over infinite *binary* trees μ-calculus and MSO are the same (Niwinski 1988).

First-Order based formalisms

- These are often studied in connection with XPath
- XPath – a W3C standard, essentially the navigation language for XML.

XPath – an informal introduction

- XPath has two kinds of formulae: node tests and path formulae.
- Node tests are closed under Boolean connectives and can check if a path satisfying a path formula can start in a given node.
- Path formulae can:
  - test if a node test is true in the first node of a path;
  - test if a path starts by going to a child, first child, next child, previous child, parent, descendant, ansector, etc;
  - take union or composition of two paths.

Example: `//book[/author[name="GW Bush"]]/title`
gives titles of books coauthored by Bush.

CTL* vs XPath

- There is a well-known logic, CTL*, that similarly combines node (called *state*) and path formulae.
- Syntax:

state formulae	$\alpha \ := \ a \mid \alpha \vee \alpha' \mid \neg\alpha \mid \mathbf{E}\beta$
path formulae	$\beta \ := \ \mathsf{LTL}$ over state formulae

Example: all descendants of a given node (including self) are labeled a (with $\Sigma = \{a, b\}$):

$$\neg\mathbf{E}\left(\ (a \vee b)\ \mathbf{U}\ b\right)$$

CTL* and FO over trees

Theorem With respect to Boolean queries:

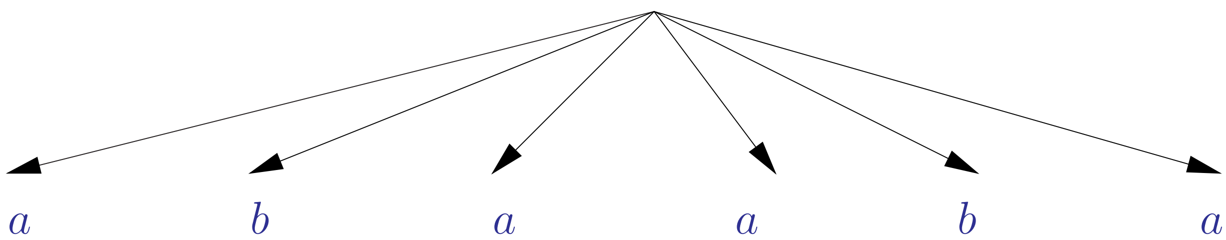
- over binary trees, CTL* $=$ FO (Hafer, Thomas, 1987).
- over unranked trees, CTL* $=$ FO (Barcelo, L., 2005; closely related to Marx 2004)
- For unary queries, one adds reasoning about the past (temporal operators $\mathbf{Y}$ – yesterday, and $\mathbf{S}$ – since).

- A technical issue: what is a path in an unranked tree? It could be a path that may change directions from siblings to children, or one could use two different kinds of path formulae.
- It turns out that this decision does not affect expressiveness.

LOGICS FOR UNORDERED TREES

Easy punchline

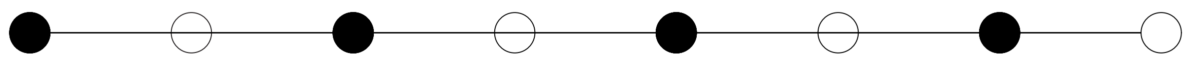
- Order buys us counting.
- Without order, counting has to be introduced explicitly.



- There is no way to say in a temporal logic that there are at least 2 children labeled a.

MSO, order, and counting

- With MSO, ordering gives us even more powerful modulo counting.
- Example: parity in MSO

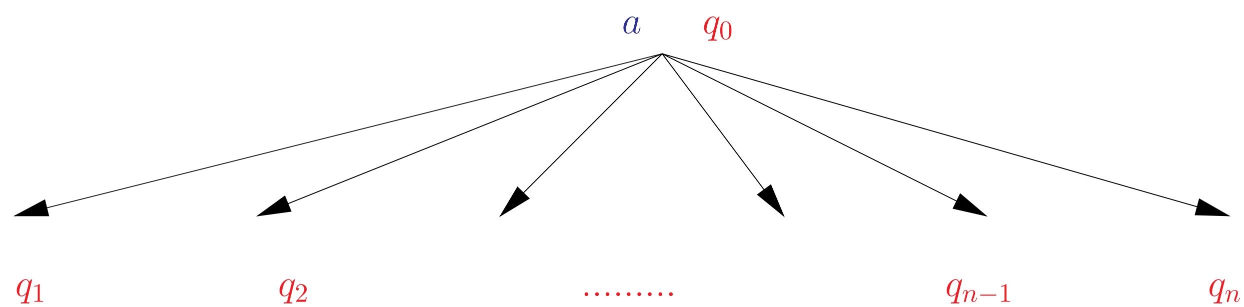


- The black set:
  - contains the first element;
  - contains every other element;
  - does not contain the last element.
- But if we only have:

we cannot say it.

Automata with counting

- New transition: δ : States $\times\ \Sigma \longrightarrow$ Boolean function over(V)
- $V \ = \ \{v_q^k \ \ | \ \ k \in \mathbb{N},\ q \in \text{States}\}$.
- A new notion of run:



- For each q, set v_q^k to true if the number of children in state q is at least k.
- If $\delta(q_0, a)$ evaluates to true, then state q_0 can be assigned.

Counting in temporal logics

Extend μ-calculus and CTL* to counting versions by changing $\mathbf{X}$ to $\mathbf{X}^k$, meaning the existence of at least k elements satisfying a formula.

Theorem For Boolean queries:

- MSO $=$ counting μ-calculus
  (Walukiewicz et al, 2002)
- FO $=$ counting CTL*
  (Moller, Rabinovich, 2003)
- For unary queries, one needs both counting and past
  (Schlingloff 1992, Barcelo, L, 2005)

Adding an arbitrary ordering

- Parity example: an order is needed, but it does not matter which one!
- Such properties are called order-invariant.

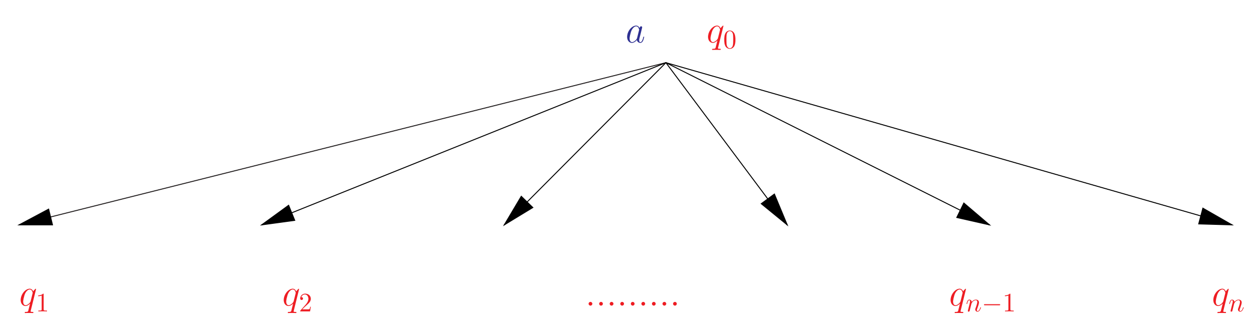
- CMSO = Counting MSO: extension of MSO with

$$\mathsf{Mod}_q(X) \qquad \text{meaning} \qquad |X| = 0 \ (\mathsf{mod}\ q)$$

Theorem (Courcelle 1991) Over trees,

$$\text{order-invariant MSO} \quad = \quad \text{CMSO}.$$

Even more powerful counting



- For each q, let v_q be the number of nodes assigned state q.
- In a more powerful counting automata, transition $\delta(q_0, a)$ could be a formula of Presburger Arithmetic (that is, $\langle \mathbb{N}, + \rangle$) over v_q's.
  For example,

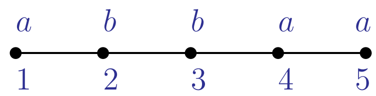
$$\delta(q_0, a) \;=\; \left(v_{q_1} + v_{q_2} = 2 \cdot v_{q_3}\right) \wedge \exists x \; \left(v_{q_4} = x + x + x\right)$$

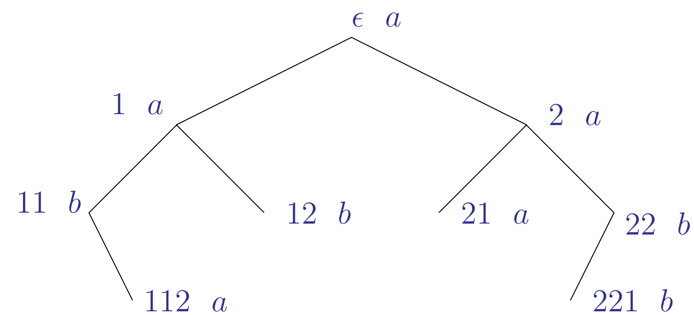
- Such automata investigated by Seidl, Schwentick, Muscholl, '03–04.
- Decidability results and fixed-point logic characterizations.

AUTOMATIC STRUCTURES

Strings, trees, and logic: a reminder

Strings and trees are viewed as finite structures.
Universe: $\{1, \ldots, n\}$ or a prefix-closed subset of $\{1, 2\}^*$
Predicates: $\prec$ – prefix, P_a and P_b for positions labeled a and b





A string s or a tree T is a structure, M_s or M_T, of vocabulary $(\prec, P_a, P_b)$.
If Φ is a sentence of a logic $\mathcal{L}$, then

$$\{s \mid M_s \models \Phi\} \qquad \{T \mid M_T \models \Phi\}$$

define string and tree languages.

Classical Results Regular languages $=$ MSO-definable

A different approach: automatic structures

If $\mathcal{M} = \langle \Sigma^*, \Omega \rangle$ is a first-order structure, then a formula $\varphi(x)$ defines the language

$$\{s \in \Sigma^* \mid \mathcal{M} \models \varphi(s)\}$$

A structure is automatic if all such languages are regular.

Operations on strings:	• $x \prec y$: x is a prefix of y • $f_a(x) = x \cdot a$, $a \in \Sigma$ • $\mathsf{el}(x, y)$ (equal length): $|x| = |y|$

$$\boxed{\mathfrak{S} \quad \stackrel{def}{=} \quad \langle\ \Sigma^*,\ \prec,\ (f_a)_{a\in\Sigma},\ \mathsf{el}\ \rangle}$$

Folklore Theorem $\mathfrak{S}$ is the universal automatic structure: relations definable by formulae $\varphi(x_1, \ldots, x_n)$ are precisely the regular relations.

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$
\begin{array}{rclllllll}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & & \\
s_3 & = & \mathsf{b} & \mathsf{b} & & \cdots & & & \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} &
\end{array}
$$

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$
\begin{array}{lclllllll}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\
s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \#
\end{array}
$$

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$
\begin{array}{rclllllll}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\
s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \# \\
 & & \Uparrow & & & & & &
\end{array}
$$

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$\begin{array}{lcccccccc}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\
s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \# \\
 & & & \Uparrow & & & & &
\end{array}$$

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$
\begin{array}{rcccccccc}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\
s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \# \\
 & & & & \Uparrow & & & &
\end{array}
$$

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$
\begin{array}{lcccccccc}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\
s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \# \\
 & & & & & & \Uparrow & &
\end{array}
$$

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$
\begin{array}{lcccccccc}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\
s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \# \\
 & & & & & & & \Uparrow &
\end{array}
$$

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$
\begin{array}{lcccccccc}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\
s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \# \\
 & & & & & & & & \Uparrow
\end{array}
$$

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$
\begin{array}{lcccccccc}
s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\
s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\
s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\
\cdots & & & & & \cdots & & & \\
s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \# \\
 & & & & & & & & \Uparrow
\end{array}
$$

The alphabet of this automaton is $(\Sigma \cup \{\#\})^n$.

Regular relations

These are n-tuples of strings accepted by letter-to-letter automata.

$$\begin{array}{lcccccccc} s_1 & = & \mathsf{a} & \mathsf{a} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{b} & \mathsf{c} \\ s_2 & = & \mathsf{a} & \mathsf{b} & \mathsf{a} & \cdots & \mathsf{a} & \# & \# \\ s_3 & = & \mathsf{b} & \mathsf{b} & \# & \cdots & \# & \# & \# \\ \cdots & & & & & \cdots & & & \\ s_n & = & \mathsf{a} & \mathsf{b} & \mathsf{b} & \cdots & \mathsf{a} & \mathsf{c} & \# \\ & & & & & & & & \Uparrow \end{array}$$

The alphabet of this automaton is $(\Sigma \cup \{\#\})^n$.

A reduct of $\mathfrak{S}$: $\boxed{\mathfrak{S}_{\mathfrak{p}} \stackrel{def}{=} \langle\ \Sigma^*,\ \prec,\ (f_a)_{a\in\Sigma}\ \rangle}$

Theorem (Benedikt, L., Schwentick, Segoufin, 2001)
Languages definable over $\mathfrak{S}_{\mathfrak{p}}$ are precisely the star-free languages.

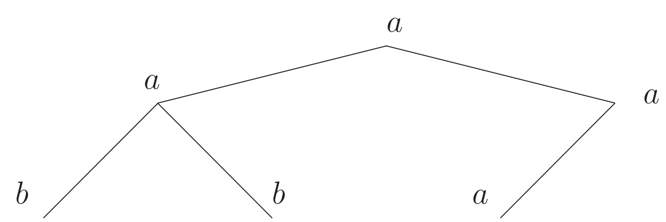
Automatic structures on binary trees

$\mathsf{Trees}(\Sigma)$ – the infinite set of all binary Σ-labeled trees.

Operations and predicates on trees:

T' extends T
(or T is subsumed by T'): $T \prec T'$

T



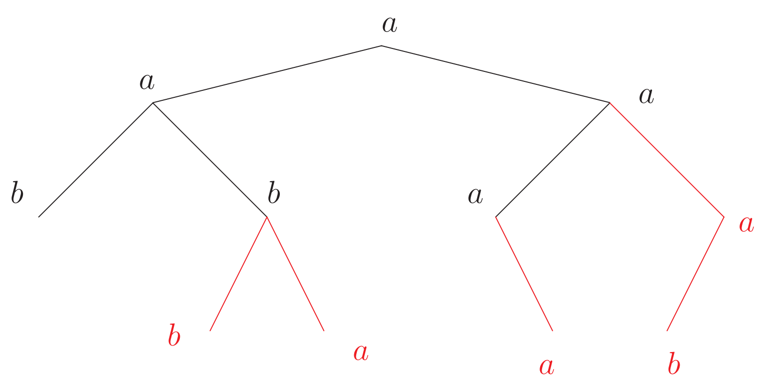
Automatic structures on binary trees

$\mathsf{Trees}(\Sigma)$ – the infinite set of all binary Σ-labeled trees.

Operations and predicates on trees:

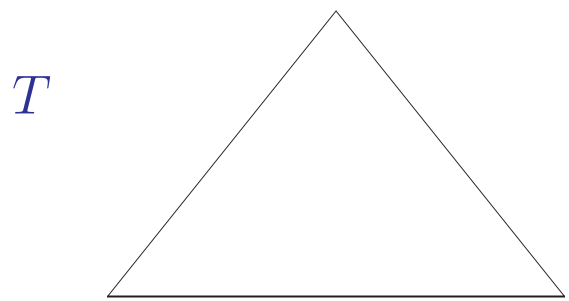
T' extends T
(or T is subsumed by T'): $\quad T \prec T'$

T'



Operations on binary trees cont'd

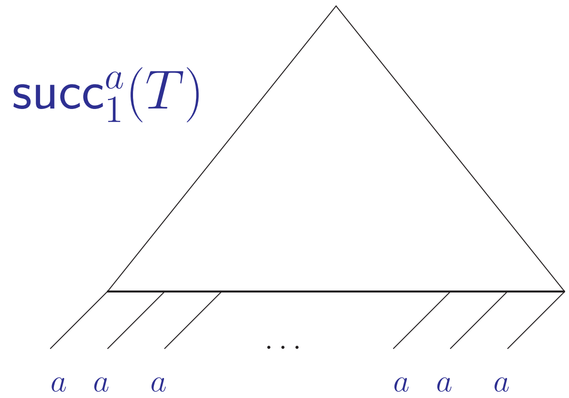
Successor operations: succ_1^a, succ_1^b, succ_2^a, succ_2^b.
succ_1^a adds, to each leaf, a left child labeled a:



Operations on binary trees cont'd

Successor operations: succ_1^a, succ_1^b, succ_2^a, succ_2^b.

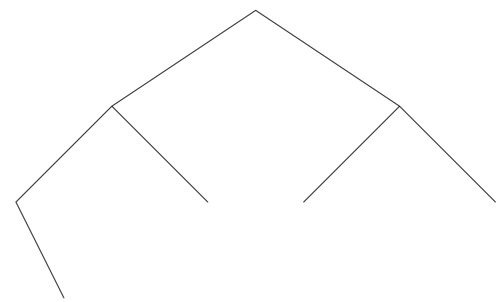
succ_1^a adds, to each leaf, a left child labeled a:



Operations on binary trees cont'd

Analog of equal length – domain equivalence:

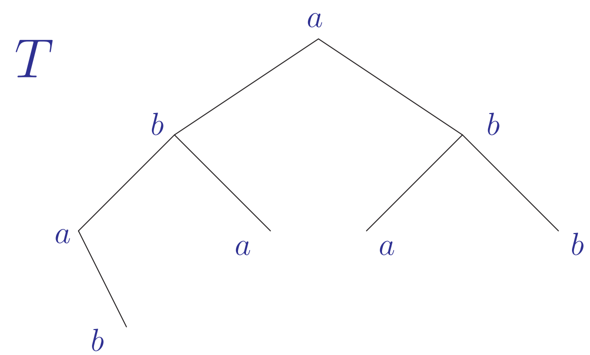
$$T \approx_{\mathsf{dom}} T' \qquad \Leftrightarrow \qquad \mathsf{domain}(T) = \mathsf{domain}(T')$$



Operations on binary trees cont'd

Analog of equal length – domain equivalence:

$$T \approx_{\mathsf{dom}} T' \quad \Leftrightarrow \quad \mathsf{domain}(T) = \mathsf{domain}(T')$$

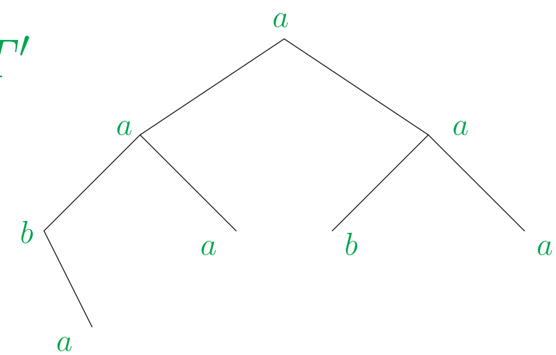


Operations on binary trees cont'd

Analog of equal length – domain equivalence:

$$T \approx_{\mathsf{dom}} T' \qquad \Leftrightarrow \qquad \mathsf{domain}(T) = \mathsf{domain}(T')$$

T'



Tree-Automatic Structures

$$\boxed{\mathfrak{T} \quad = \quad \langle \ \mathsf{Trees}(\Sigma), \ \prec, \ \mathsf{succ}^{a,b}_{1,2}, \ \approx_{\mathsf{dom}} \ \rangle}$$

Tree-Automatic Structures

$$\boxed{\mathfrak{T} \;=\; \langle\ \mathsf{Trees}(\Sigma),\ \prec,\ \mathsf{succ}_{1,2}^{a,b},\ \approx_{\mathsf{dom}}\ \rangle}$$

$$\boxed{\mathfrak{T}_{\mathfrak{p}} \;=\; \langle\ \mathsf{Trees}(\Sigma),\ \prec,\ \mathsf{succ}_{1,2}^{a,b}\ \rangle}$$

Tree-Automatic Structures

$$\boxed{\mathfrak{T} \;=\; \langle\ \mathsf{Trees}(\Sigma),\ \prec,\ \mathsf{succ}_{1,2}^{a,b},\ \approx_{\mathsf{dom}}\ \rangle}$$

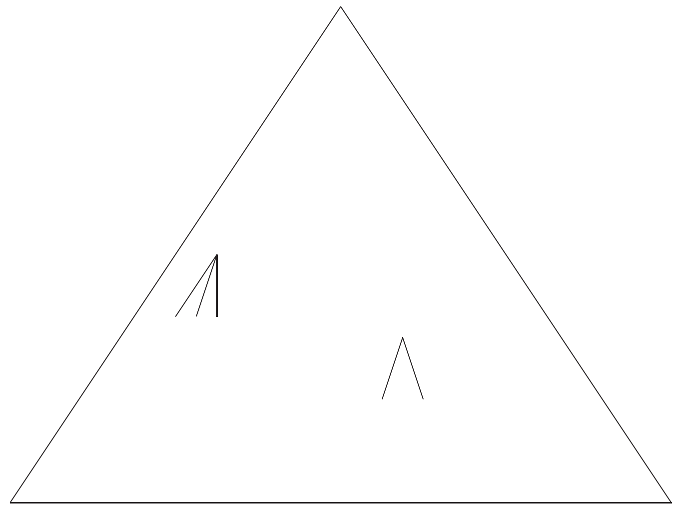
$$\boxed{\mathfrak{T}_{\mathfrak{p}} \;=\; \langle\ \mathsf{Trees}(\Sigma),\ \prec,\ \mathsf{succ}_{1,2}^{a,b}\ \rangle}$$

Theorem (Benedikt, L., 2002)

- For both $\mathfrak{T}_{\mathfrak{p}}$ and $\mathfrak{T}$, the class of definable sets is precisely the class of regular tree languages.
- $\mathfrak{T}$ is the universal tree-automatic structure: a relation on $\mathsf{Trees}(\Sigma)$ is $\mathfrak{T}$-definable iff it is regular.
- $\mathfrak{T}_{\mathfrak{p}}$ is weaker than $\mathfrak{T}$.

Operations on unranked trees

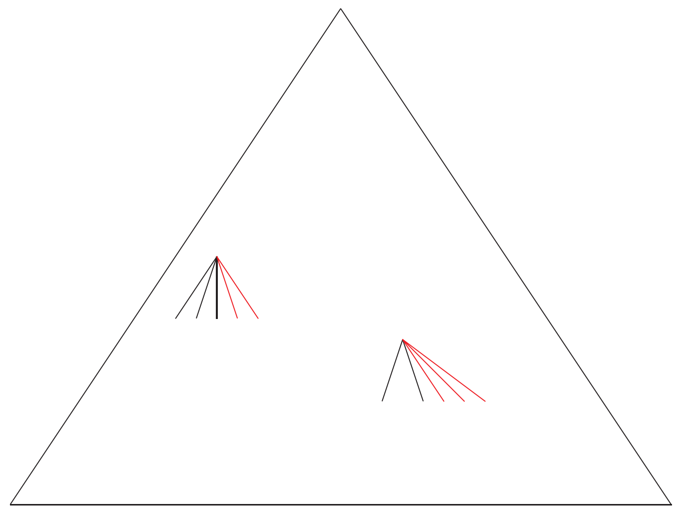
Reusing the extension operation $\prec$ requires infinitely many successor operations, which is undesirable. Hence, we split it into two: extension right $\prec_{\rightarrow}$ and extension down $\prec_{\downarrow}$.



$$T \prec_{\rightarrow} T'$$

Operations on unranked trees

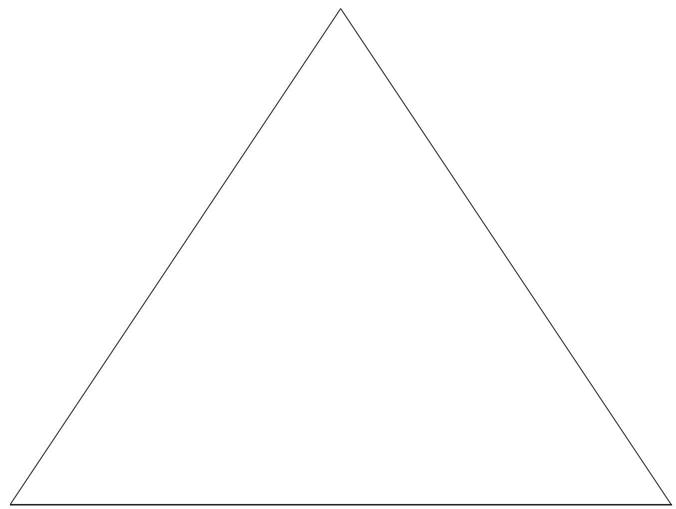
Reusing the extension operation $\prec$ requires infinitely many successor operations, which is undesirable. Hence, we split it into two: extension right $\prec_{\rightarrow}$ and extension down $\prec_{\downarrow}$.



$$T \prec_{\rightarrow} T'$$

Operations on unranked trees

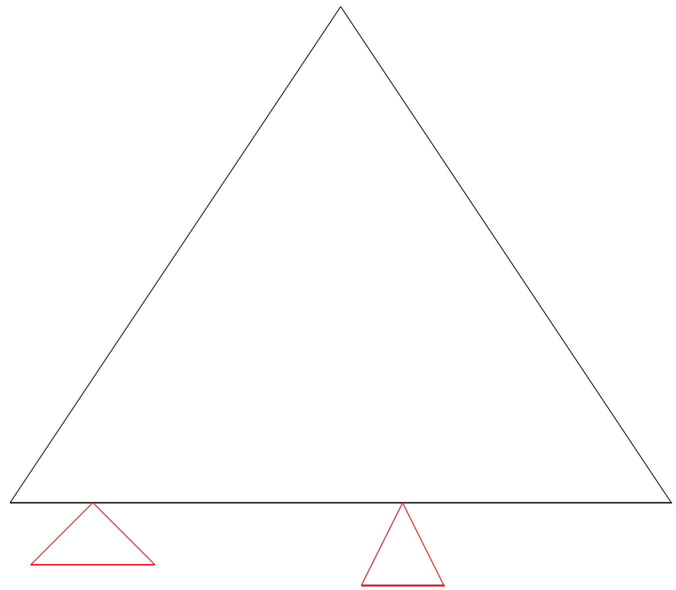
Reusing the extension operation $\prec$ requires infinitely many successor operations, which is undesirable. Hence, we split it into two: extension right $\prec_\rightarrow$ and extension down $\prec_\downarrow$.



$T \prec_\downarrow T'$

Operations on unranked trees

Reusing the extension operation $\prec$ requires infinitely many successor operations, which is undesirable. Hence, we split it into two: extension right $\prec_{\rightarrow}$ and extension down $\prec_{\downarrow}$.



$T \prec_{\downarrow} T'$

Unranked Tree-Automatic Structures

$$\boxed{\mathfrak{T}^{\mathfrak{u}} \quad = \quad \langle\ \mathsf{UTrees}(\Sigma),\ \prec_{\rightarrow},\ \prec_{\downarrow},\ (L_a)_{a\in\Sigma},\ \approx_{\mathsf{dom}}\ \rangle}$$

Here $L_a(T)$ is true if the rightmost node of T is labeled a.

Unranked Tree-Automatic Structures

$$\boxed{\mathfrak{T}^{\mathfrak{u}} \quad = \quad \langle\ \mathsf{UTrees}(\Sigma),\ \prec_{\rightarrow},\ \prec_{\downarrow},\ (L_a)_{a\in\Sigma},\ \approx_{\mathsf{dom}}\ \rangle}$$

Here $L_a(T)$ is true if the rightmost node of T is labeled a.

$$\boxed{\mathfrak{T}^{\mathfrak{u}}_{\mathfrak{p}} \quad = \quad \langle\ \mathsf{UTrees}(\Sigma),\ \prec_{\rightarrow},\ \prec_{\downarrow},\ (L_a)_{a\in\Sigma}\ \rangle}$$

Unranked Tree-Automatic Structures

$$\boxed{\mathfrak{T}^{\mathfrak{u}} \quad = \quad \langle\ \mathsf{UTrees}(\Sigma),\ \prec_{\rightarrow},\ \prec_{\downarrow},\ (L_a)_{a\in\Sigma},\ \approx_{\mathsf{dom}}\ \rangle}$$

Here $L_a(T)$ is true if the rightmost node of T is labeled a.

$$\boxed{\mathfrak{T}^{\mathfrak{u}}_{\mathfrak{p}} \quad = \quad \langle\ \mathsf{UTrees}(\Sigma),\ \prec_{\rightarrow},\ \prec_{\downarrow},\ (L_a)_{a\in\Sigma}\ \rangle}$$

Unranked Tree-Automatic Structures: basic results

Theorem (L., Neven, 2003)

- Unranked tree languages definable in $\mathfrak{T}^{\mathfrak{u}}$ and $\mathfrak{T}^{\mathfrak{u}}_{\mathfrak{p}}$ are precisely the regular unranked tree languages.

Unranked Tree-Automatic Structures: basic results

Theorem (L., Neven, 2003)

- Unranked tree languages definable in $\mathfrak{T}^{\mathfrak{u}}$ and $\mathfrak{T}^{\mathfrak{u}}_{\mathfrak{p}}$ are precisely the regular unranked tree languages.
- $\mathfrak{T}^{\mathfrak{u}}$ is the universal unranked tree automatic structure: relations definable in $\mathfrak{T}^{\mathfrak{u}}$ are precisely the regular unranked tree relations.

Unranked Tree-Automatic Structures: basic results

Theorem (L., Neven, 2003)

- Unranked tree languages definable in $\mathfrak{T}^{\mathfrak{u}}$ and $\mathfrak{T}^{\mathfrak{u}}_{\mathfrak{p}}$ are precisely the regular unranked tree languages.
- $\mathfrak{T}^{\mathfrak{u}}$ is the universal unranked tree automatic structure: relations definable in $\mathfrak{T}^{\mathfrak{u}}$ are precisely the regular unranked tree relations.
- The theory of $\mathfrak{T}^{\mathfrak{u}}$ is decidable.

Unranked Tree-Automatic Structures: basic results

Theorem (L., Neven, 2003)

- Unranked tree languages definable in $\mathfrak{T}^{\mathfrak{u}}$ and $\mathfrak{T}^{\mathfrak{u}}_{\mathfrak{p}}$ are precisely the regular unranked tree languages.
- $\mathfrak{T}^{\mathfrak{u}}$ is the universal unranked tree automatic structure: relations definable in $\mathfrak{T}^{\mathfrak{u}}$ are precisely the regular unranked tree relations.
- The theory of $\mathfrak{T}^{\mathfrak{u}}$ is decidable.
- $\mathfrak{T}^{\mathfrak{u}}_{\mathfrak{p}}$ is weaker than $\mathfrak{T}^{\mathfrak{u}}$.

Ranked an unranked branches

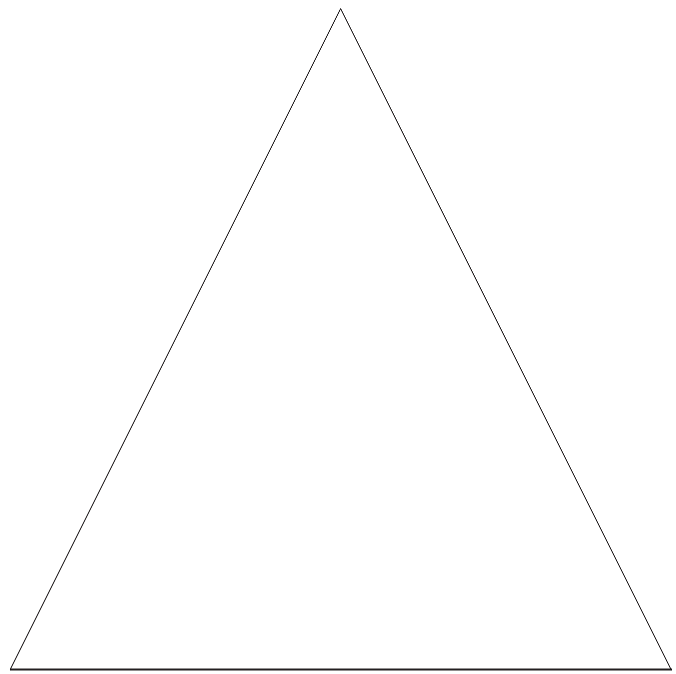
A tree T is a branch if

$$\forall T',\ T'' \preceq T\ \Big(\ (T' \preceq T'')\ \vee\ (T'' \preceq T')\ \Big)$$

Ranked an unranked branches

A tree T is a branch if

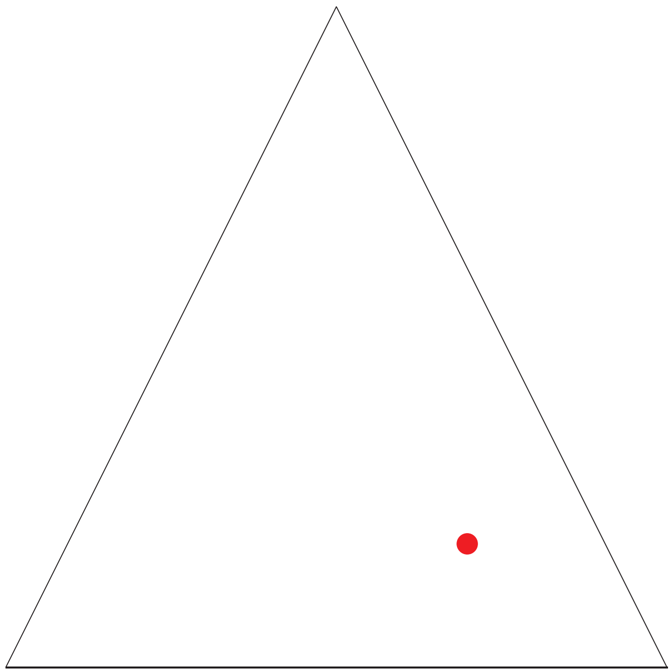
$$\forall T', \; T'' \preceq T \; \Big(\, (T' \preceq T'') \; \vee \; (T'' \preceq T') \, \Big)$$



Ranked an unranked branches

A tree T is a branch if

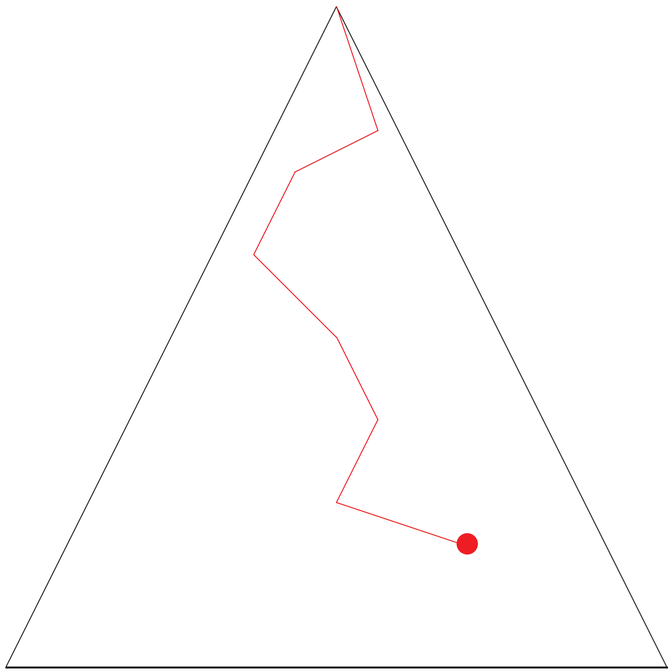
$$\forall T', \, T'' \preceq T \left(\, (T' \preceq T'') \, \vee \, (T'' \preceq T') \, \right)$$



Ranked an unranked branches

A tree T is a branch if

$$\forall T', \, T'' \preceq T \left(\, (T' \preceq T'') \, \vee \, (T'' \preceq T') \, \right)$$

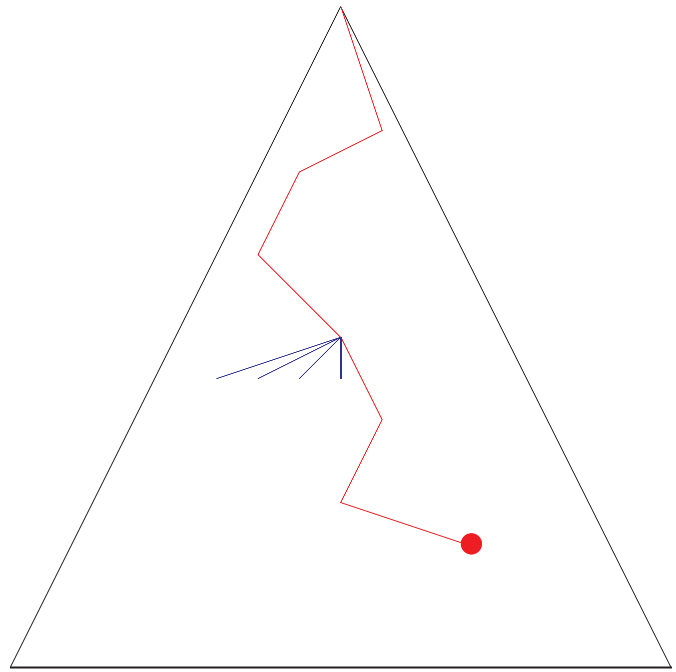


Ranked branch

Ranked an unranked branches

A tree T is a branch if

$$\forall T', \; T'' \preceq T \; \Big(\; (T' \preceq T'') \; \vee \; (T'' \preceq T') \; \Big)$$



Unranked branch

Logics with branch quantification

We write **FO**$_\eta$ to indicate that we quantify only over branches.

Then definable sets of trees have analog in the classical theory of logical definability over trees, which uses logics such as $\mathcal{FO}$, $\mathcal{MSO}$, $\mathcal{MSO}^{path}$ (only quantification over chains is allowed).

Logics with branch quantification

We write $\mathbf{FO}_\eta$ to indicate that we quantify only over branches.

Then definable sets of trees have analog in the classical theory of logical definability over trees, which uses logics such as $\mathcal{FO}$, $\mathcal{MSO}$, $\mathcal{MSO}^{path}$ (only quantification over chains is allowed).

Theorem Over ranked trees:

$$\mathbf{FO}_\eta(\mathfrak{T}_\mathfrak{p})\text{-definable} = \mathcal{FO}\text{-definable}$$

$$\mathbf{FO}_\eta(\mathfrak{T})\text{-definable} = \mathcal{MSO}^{path}\text{-definable}$$

Logics with branch quantification

We write $\mathbf{FO}_\eta$ to indicate that we quantify only over branches.

Then definable sets of trees have analogs in the classical theory of logical definability over trees, which uses logics such as $\mathcal{FO}$, $\mathcal{MSO}$, $\mathcal{MSO}^{path}$ (only quantification over chains is allowed).

Theorem Over ranked trees:

$$\mathbf{FO}_\eta(\mathfrak{T}_\mathfrak{p})\text{-definable} = \mathcal{FO}\text{-definable}$$
$$\mathbf{FO}_\eta(\mathfrak{T})\text{-definable} = \mathcal{MSO}^{path}\text{-definable}$$

Over unranked trees:

$$\mathbf{FO}_\eta(\mathfrak{T}_\mathfrak{p}^\mathfrak{u})\text{-definable} = \mathcal{FO}\text{-definable}$$
$$\mathbf{FO}_\eta(\mathfrak{T}^\mathfrak{u})\text{-definable} = \mathcal{MSO}^{\uparrow}_{\rightarrow}\text{-definable}$$

$\mathcal{MSO}^{\uparrow}_{\rightarrow}$ is $\mathcal{MSO}$ with quantification restricted to vertical and horizontal paths: an analog of $\mathcal{MSO}^{path}$ for unranked trees.

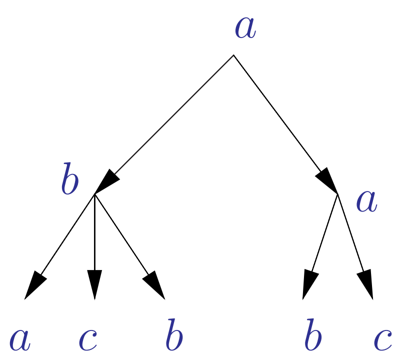
What else is in the survey?

- Edge-labeled trees.
- They occur in a variety of areas:
  - computational linguisticts;
  - ambient and spatial logics.
- Logics have quite a different flavor.
- Connections between them and logics considered here are being explored.

Other directions

- We have seen plenty of declarative specification languages with good associated procedural formalisms in terms of model-checking properties.
- What causes them to be good?
- One way to look at this: succinctness (Grohe/Schweikardt).
  How big are formulae for expressing certain properties?

Other directions: streaming




```
<a>
 <b>
  <a></a>
  <c></c>
  <b></b>
 </b>
 <a>
  <b></b>
  <c></c>
 </a>
</a>
```

Streamed representation:

$$aba\bar{a}c\bar{c}b\bar{b}\bar{b}ab\bar{b}c\bar{c}\bar{a}\bar{a}$$

Question: what properties of trees can we check by a finite automaton over the streamed representation?

Since the language of balanced parentheses is not regular, we may assume the input is already a valid stream.

Other directions: streaming cont'd

- Example The following DTD is not stream-verifiable (Segoufin/Vianu 2002):
  $a \rightarrow ab \mid ca \mid \varepsilon$
  $b \rightarrow \varepsilon$
  $c \rightarrow \varepsilon$
- Originally an involved pumping-lemma argument, but logic gives a much simpler proof:
- For every MSO sentence φ one can find two strings of the form

$$ab\bar{b}ab\bar{b}\ldots ab\bar{b}a\ldots a\bar{a}c\bar{c}\ldots\bar{a}c\bar{c}\bar{a}\ldots\bar{a}$$

  that agree on φ; one of them corresponds to the above DTD, and the other one to:
  $a \rightarrow a \mid ab \mid ca \mid \varepsilon$
  $b \rightarrow \varepsilon$
  $c \rightarrow \varepsilon$

Other directions: streaming cont'd

- There is a characterization of a fragment of MSO over trees that defines precisely the "streamable" properties (checked by string automata).
- However, decidability of that fragment remains open.

Other directions: data values

- So far we considered only labels on trees (e.g., book, author) but no data values (e.g., "WH Press").
- Adding data values quickly leads to undecidability.
- Example: DTDs + key/foreign key constraints.
  - Satisfiability problem: is a specification consistent?
  - Some known results (Fan, L., 2001):
  - It is NP-complete for unary constraints (e.g. title determines publisher).
  - It is undecidable even for binary constraints (e.g., title and author determine publisher).

Other directions: data values

- Proofs were not logic-based (mostly combinatorial plus integer linear programming).
- But it appears that logic can provide an explanation.
- Consider strings with data values attached to positions.
- Bojanczyk, Muscholl, Schwentick, Segoufin, 2005:
  - FO^2, first-order with two variables, is decidable.
  - FO^3, first-order with three variables, is undecidable.
- One needs two variables to talk about unary constraints, and more for binary, etc., constraints.

Summing up

- XML application give theoreticians nice problems to work on.
- Combination of well developed tools:
  - formal languages,
  - logic,
  - string and tree automata
- Not everything is a straightforward adaption of old and known results.